HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80528-9599

02-03-00

A

PATENT APPLICATION

ATTORNEY DOCKET NO. 10971393-1

JC760 U.S. PTO
02/02/00

JC530 U.S. PTO
09/496844
02/02/00

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): (X) Utility ( ) Design
(X) original patent application,
( ) continuation-in-part application

INVENTOR(S): **Patrick Knebel et al**

TITLE: **Method And Apparatus For Emulating SIMD Instructions**

Enclosed are:

(X) The Declaration and Power of Attorney. (X) signed ( ) unsigned or partially signed
(X) 5 sheets of drawings (one set) ( ) Associate Power of Attorney
( ) Form PTO-1449 ( ) Information Disclosure Statement and Form PTO-1449
( ) Priority document(s) ( )(Other) ____________ (fee $ ______)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	18 — 20	0	X $18	$ 0
INDEPENDENT CLAIMS	3 — 3	0	X $78	$ 0
ANY MULTIPLE DEPENDENT CLAIMS	0		$260	$ 0
BASIC FEE: Design $310.00); Utility $(690.00)				$ 690
TOTAL FILING FEE				$ 690
OTHER FEES				$
TOTAL CHARGES TO DEPOSIT ACCOUNT				$ 690

Charge $ 690 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16, 1.17, 1.19, 1.20 and 1.21. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EL523589306US

Date of Deposit 2/2/00

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

By *Linda C. Cunningham*

Typed Name: **Linda C. Cunningham**

Respectfully submitted,

Patrick Knebel et al

By ____________

Alexander J Neudeck

Attorney/Agent for Applicant(s)
Reg. No. 41220
Date: 2-1-00
Telephone No.: **(970) 898-4931**

METHOD AND APPARATUS FOR EMULATING SIMD INSTRUCTIONS

I. FIELD

The present invention relates to digital computer systems, and more particularly but not by way of limitation, to methods and an apparatus for processing instructions in such systems.

II. BACKGROUND

Microprocessors exist that implement a reduced instruction set computing (RISC) instruction set architecture (ISA) and an independent complex instruction set computing (CISC) ISA by emulating the CISC instructions with instructions native to the RISC instruction set. Instructions from the CISC ISA are called "macroinstructions." Instructions from the RISC ISA are called "microinstructions."

Streaming Single-Instruction Multiple-Data Extensions (SSEs) have been developed to enhance the instruction set of the latest generation of certain computer architectures, for example the IA-32 architecture. The SSEs include a new set of registers, new floating point data types, and new instructions. Specifically, the SSEs comprise eight 128-bit Single-Instruction Multiple-Data (SIMD) floating point registers (XMM0 through XMM7) that can be used to perform calculations and operations on floating point data. These XMM registers are shown in Figure 1A. Each 128-bit floating point register contains four packed 32-bit single precision (SP) floating point (FP) numbers. The structure of the packed 32-bit SP FP numbers is illustrated in the example of Figure 1B, where four 32-bit SP FP numbers (numbered 0 through 3) are shown as if stored in the XMM2 SSE register. In architectures designed to support the SSEs, such as its native architecture, a single instruction of the SSE instruction set operates in parallel on the four 32-bit SP FP numbers in a particular XMM register.

The SSEs also include a status and control register called the MXCSR register. The format of the MXCSR is illustrated in the example of Figure 1C. The MXCSR register may be used to selectively mask or unmask exceptions. Specifically, bits 7-12 of the MXCSR register may be used by a programmer to selectively mask or unmask a particular exception. Masked exceptions are those exceptions that a programmer wishes to be handled automatically by the processor which may provide a default response. Unmasked exceptions, on the other hand, are

those exceptions that the programmer wishes to be handled by invocation of an interrupt or an operating system handler. This invocation of the handler transfers control of the operating system (such as Windows by Microsoft) where the problem may be corrected or the program terminated.

The MXCSR register may also be used to keep track of the status of exception flags. Bits 0-5 of the MXCSR register indicate whether any of six exceptions have occurred in the execution of an SSE instruction. Those exceptions include the following: invalid operation (I), divide-by-zero (Z), denormal operation (D), numeric overflow (O), numeric underflow (U), or inexact result (P). The status of the flags are "sticky," meaning that once they are set, they are not cleared by any subsequent SSE instruction, even if one is performed without exception. The status flags can only be cleared by a special instruction, usually issued from the operating system.

The exception flags of Figure 1C are the result of a bitwise logical-OR operation on all four of the 32-bit SP FP operations that are performed on a particular 128-bit register XMM register. (One operation on each of the four 32-bit SP FP numbers.) Thus, if an exception occurs as to any one of the four 32-bit SP FP numbers, the exception flag for that particular type of exception will be raised, indicating some type of problem has occurred in the system. The invalid operation (I) divide by zero (Z), and denormal operation (D) exceptions are pre-computation exceptions, meaning that they are detected before any arithmetic or logical operations occur. That is, they can be detected without doing any computations. The other three exceptions, numeric overflow (O), numeric underflow (U), and inexact result (P) are post-computation exceptions, meaning that they are detected after the operations have been performed. It is possible for an operation performed on a sub-operand (i.e., one of the four operands in a 128-bit XMM register) to raise multiple flags.

SSEs have the following rules for exceptions:

1. When an unmasked exception occurs, the processor executing the instruction will not change the contents of the XMM register. In other words, results will not be committed or stored until it is known that no unmasked exceptions have occurred with respect to any of the four 32-bit SP FP numbers.

2. If there is a masked exception, all exception flags are updated.

3. In the case of unmasked pre-computation exceptions, all flags relating to pre-computation exceptions, whether masked or unmasked, will be updated. However, no subsequent computations are permitted, meaning that no post-execution exceptions can or will occur. This, of course, means that no post-execution exception flags will change or be updated.

4. In the case of unmasked post-computation exceptions, all post-execution conditions, whether masked or unmasked, will be updated, as will all pre-computation exceptions. Any pre-computation exceptions will be masked exceptions only because, if the pre-computation exception was unmasked, under Rule No. 3 above, no further computations would have been permitted.

Further information regarding streaming SIMD extensions may be found in the Intel Architecture Software Developer's Manual, (1999), Volumes 1 through 3, Intel Order Numbers 243190, 243191, 243192, which are hereby incorporated by reference.

In many architectures, provisions have not been made for the SSE instructions. In these non-native architectures, the eight 128-bit floating point XMM registers capable of containing four 32-bit SP FP numbers are not available. In some non-native architectures, the eight 128-bit XMM registers may be mapped onto 16 floating point registers (e.g., IA-64 registers) that may be less than 128 bits and more than 64 bits wide. Specifically, some architectures use 82-bit registers to hold two 32-bit SP FP numbers (the bits in excess of 64 may be used for the special encoding used to indicate the register holes SIMD-type 32-bit SP FP numbers). An example is shown in Figure 1D. Note that the four 32-bit SP FP numbers 0-3 stored in the XMM2 register of the SSE native environment (Figure 1B) are now stored in two 82-bit registers, XMM2_Low and XMM2_High, containing the "low half of the XMM2 register" and the "high half of the XMM2 register," respectively. This makes parallel execution of an operation on each of the four 32-bit SP FP numbers difficult.

Thus, in this non-native environment, the SSE instructions must be executed by emulation. Specifically, operations may first be performed on two of the four 32-bit SP FP numbers (in parallel) and then be performed on the remaining two 32-bit SP FP numbers (again, in parallel). Operations may alternatively be performed on only one or at least three of the 32-bit SP FP numbers. For example, an operation

may be performed on the operands in the low half, XMM2_Low, and then on the high half, XMM2_High. However, given the SSE rules for handling exceptions and updating exception flags, problems arise when emulating SSE instructions in this partially parallel, partially sequential manner. For example, consider a set of instructions being performed on the low half and high half of Figure 1D:

XMM2 := OP(XMM3, XMM4)

emulated by

XMM2_Low := OP(XMM3_Low, XMM4_Low)

XMM2_High := OP(XMM3_High, XMM4_High)

Assume that the first instruction is executed without an unmasked exception as to the operands in the low halves, XMM3_Low and XMM4_Low. The results of this operation are then properly committed in XMM2_Low. Assume now that execution of the second instruction on the high halves results in a pre-computation unmasked exception. According to the SSE rules, no subsequent operations are then to be performed on any of the four 32-bit SP FP numbers because of that pre-computation unmasked exception. But here, however, results of the operation on the low halves have been committed to register XMM2_Low in violation of the SSE rules. This corrupts the data in XMM2_Low and cannot be allowed to happen.

Prior machines have solved this problem by implementing a "back-off" mechanism that allowed them to speculatively change architectural state when the first microinstruction completed, then "undo" the change if the second microinstruction had an exception. This back-off mechanism may be hard to implement in certain machines, especially in machines that do not implement register renaming. In many systems, the use of a back-off or undo operation is difficult or limited, for various reasons.

One way of successfully emulating the SSEs and preventing this rule violation is to use a "shadow" register mechanism. In a shadow register mechanism, the results of a previous, successful operation on the low halves are physically stored in a shadow register. In this case, in the example above, when the exception is detected on the high halves, the results previously stored in the shadow register for the previous operation on the low halves may be re-stored, that is, an "undo" operation on the low halves as performed. The shadow register mechanism, however, is

relatively complex. In most systems, there must be at least 16 registers available for storing the results of a previous operation on the low halves, and each must be capable of two 32-bit FP SP numbers. Additionally, when an "undo" operation is required, it must be determined which of the shadow registers the desired results are in. This mechanism consumes valuable register space that could otherwise be used more efficiently. Furthermore, a relatively complicated system of pointers and virtual maps are required to store the previous maps.

Another way to emulate a particular SSE instruction is to provide a back-off register mechanism. One skilled in the art will recognize that this technique may require a plurality of registers, a multiplexor and a de-multiplexor combination, various other hardware, and a new set of instructions. All of these increase cost and reduce efficiency.

Yet another way to emulate a particular SSE instruction is to execute the instruction with respect to each of the four 32-bit SP FP numbers in the SSE XMM register one at a time and store the results of each execution in temporary registers. When the instruction has been executed with respect to the fourth 32-bit SP FP number, and no unmasked exceptions have occurred, the results may then be committed to the appropriate architectural location and exception flags be updated. This method of emulation requires the addition of a relatively complex micro-code sequence and the use of hardware that could otherwise be used more efficiently, not to mention the amount of clock cycles it consumes in executing an instruction four times before results can be committed.

Clearly, there exists a need for a method and an apparatus for emulating the SSE instruction set (and other instruction sets) that makes efficient use of existing hardware and that consumes relatively few clock cycles. Additionally, there exists a need for a method and apparatus for determining whether certain problems may occur in the execution of a series of instructions without committing the results of those instructions.

III. SUMMARY

The present invention is a method for processing instructions by decomposing a macroinstruction into at least two microinstructions, executing the microinstructions in parallel on two separate functional units, and linking the microinstructions such

that they appear as though they were executed as a single functional unit. The present invention operates by determining whether certain exceptions occur in either of the functional units, according to SSE rules for exceptions. If an exception does occur in any of the linked microinstructions, then the execution of each of those microinstructions is canceled. This avoids the necessity of a backoff or undo mechanism.

The present invention is also a computer system for processing software instructions, having a processor with a floating point unit, a ROM, and floating point registers. The processor is configured to decompose a macroinstruction into at least two microinstruction, to execute those microinstructions in parallel, and to link those instructions such that they appear to execute as a single functional unit. The processor also is capable of identifying and treating exceptions without the use of a back-off or undo mechanism.

IV. BRIEF DESCRIPTIONS OF THE DRAWINGS

Figures 1A through 1D are block diagrams of components of the SSEs.

Figure 2 is a block diagram of a computer system including the present invention.

Figure 3 is a block diagram of the processor in Figure 2.

Figure 4 is a flow chart of the method of the present invention.

V. DETAILED DESCRIPTION

A. The Computer System

Figure 2 illustrates a computer system 10 in which the present invention may be implemented. The computer system 10 comprises at least one processor 20, main memory 30, and various interconnecting data, address, and control busses (numbered collectively as 40). An instruction set 50, which may include SSEs, and an operating system 60 may be stored in main memory 30.

As illustrated in Figure 3, the processor 20 comprises a floating point unit 70, a micro-code ROM 100, various busses and interconnections (numbered collectively as 110) and a register file 120 comprising the sixteen floating point registers, XMM0_Low through XMM7_High, needed to emulate the SSE XMM registers. In one embodiment, the sixteen floating point registers are 82-bit registers, but other widths (e.g., 128-bit or 64-bit) may be used, as the following description in terms of

82-bit registers is exemplary only, and not intended in a limiting sense. The four 32-bit SP floating point numbers of the SSEs may be stored in two of the 82-bit SP FP registers of the present invention (e.g., XMM2_Low and XMM2_High, as illustrated in Figure 1D). The floating point unit 70 comprises a first 32-bit register 130 that corresponds to the MXCSR register of the SSEs and at least two functional units (e.g., 170, 172) used to execute two FP operations.

Instructions are provided to the processor 20 from main memory 30. The instructions provided to the processor 20 are macro-code instructions that map to one or more micro-code instructions 140 stored in the micro-code ROM 100. The micro-code instructions can be directly executed by the processor 20. Also stored in the micro-code ROM 100 are a set of micro-code handlers 150 that may be invoked to handle certain unmasked processor exceptions. The processor 20 may have a pipelined architecture and may allow for parallel processing of certain instructions.

B. In Use

The IA-32 ISA defines streaming SIMD extensions, or SSE instructions, that allow a single instruction to operate simultaneously on multiple single-precision (SP) floating-point (FP) values held in a register file that is eight entries deep and 128 bits wide. Since single-precision floating-point numbers can be represented in 32 bits, each SSE register can hold four packed SP values, and each SSE instruction can calculate four SP results. Even though the IA-32 ISA specifies 128-bit registers and instructions that can operate on 128 bits of data, an implementation may choose to implement narrower registers or FP functional units that can only calculate one or two FP SP operations at a time. It is possible, then, to implement the SSE instructions by emulating them on multiple functional units by executing the four FP SP operations serially over multiple cycles, or by some combination of both of these techniques. The ISA only requires that any architecturally visible side effects of the operations appear as though all four operations occurred concurrently in the hardware.

In a preferred embodiment, a RISC ISA is implemented containing FP registers that are 82 bits wide. Thus, to emulate SSE instructions on this implementation, two FP registers are required to emulate the requisite four 32-bit SP FP values contained in one SSE register. Also, the RISC ISA defines its own SIMD operations that operate on two SP values with one instruction. So in this

implementation, two RISC microinstructions are required to emulate a single SSE macroinstruction.

SSE instructions may cause exceptions when they execute, depending on the values of the operands. The IA-32 architecture defines six possible exceptions, three pre-computation and three post-computation. Pre-computation exceptions are those which are calculated based on the values of the source registers, whereas post-computation exceptions are calculated based on the value of the result of the computation. The architecture also defines a control and status register, the MXCSR. The MXCSR contains control bits that can independently mask each exception and flag bits that capture the status of any exceptions that occur.

A preferred embodiment of the present invention includes a method for issuing two RISC microinstructions in parallel, a method for updating the flags based on the results of both FP units, and a method for causing an exception in either FP unit to inhibit setting a result register in both FP units.

When executing in native mode (RISC ISA), the implementation is required to have precise exceptions. That is, if an instruction causes an exception, then all subsequent instructions in the instruction stream must be flushed from the process. Even if an implementation is executing multiple instructions in parallel, the sequential semantics must be adhered to. For example, in the following instruction stream:

op1 - oldest instruction
op2
op3
op4
. . .
opN - youngest instruction

Op1 is the oldest instruction, and if it takes a fault, the processor must flush all younger operations (op2 - opN). If the processor executes two operations in parallel and pipelines new operations every cycle, all these operations might be "in flight" at the same time. For example, op1 and op2 might have issued in parallel and may be close to completion, while op3 and op4 are in flight one cycle behind, op5 and op6 are in flight two cycles behind, etc. If op1 takes an exception, all operations (op1

through op8) must be flushed. But if op2 takes an exception, only operations op2 through op8 must be flushed.

Figure 4 is a flow chart showing the steps required to implement the present invention, to emulate SSE instructions. First, a macroinstruction is decomposed into two or more microinstructions in a decompose function 310. The two microinstructions used to emulate the high half and the low half of the SSE operation are forced to issue in parallel and are dispatched simultaneously to the two FP units each capable of calculating two SP values per cycle, as shown in the emulation function 320. This is accomplished via a mechanism claimed in copending patent application, entitled "Method and Apparatus for Implementing Two Architectures on a Chip," inventors Knebel, et al., filed on the same date as this application, which is hereby incorporated by reference. Both micro-operations must be treated as a single operation. They move in lockstep with each other.

A signal is then generated by the emulation hardware and sent to the FP functional unit to specify when an SSE instruction is being emulated, as shown in the signal generation function 330. Then, when a signal fires, an exception detection function 340 determines whether an exception is taken in either FP functional unit. If an exception is taken in either FP functional unit, then the results from both FP functional units must be cleared, the execution is canceled, and a handler is invoked, as shown by the cancellation function 350. The invention must be capable of flushing the result in the other FP functional units, regardless of the relative age of the microinstructions in the two functional units. When a signal above fires, the MXCSR flag must be updated based on the results of both functional units, as shown in the update function 360. Following the successful completion of the execution of the microinstructions, the process continues, as shown in the continuation function 370.

Although the present invention has been described in detail with reference to certain embodiments thereof, variations are possible. Therefore, the present invention may be embodied in other specific forms without departing from the essential spirit or attributes thereof. It is desired that the embodiments described herein be considered in all respects as illustrative, not restrictive, and that reference be made to the appended claims for determining the scope of the invention.

CLAIMS

We claim:

1. A method for processing software instructions comprising,

(a) decomposing a macroinstruction into a plurality of microinstructions,

(b) issuing at least two of the plurality of microinstructions in parallel,

(c) determining whether an exception occurs in any of the at least two of a plurality of microinstructions, and

(d) if an exception occurs in any of the at least two of a plurality of microinstructions, canceling the at least two of a plurality of microinstructions.

2. The method of claim 1, further comprising executing the at least two of the plurality of microinstructions.

3. The method of claim 2, wherein the at least two of a plurality of microinstructions are executed on separate execution units, but appear as though they were executed on a single execution unit.

4. The method of claim 1, wherein the at least two of a plurality of microinstructions are executed on the same clock cycle.

5. The method of claim 1, wherein the at least two of a plurality of microinstructions are executed over multiple clock cycles.

6. The method of claim 1, wherein the method is implemented in a system emulating SSE instructions.

7. The method of claim 6, wherein the system allows a single instruction to operate on multiple single-precision ("SP") floating-point ("FP") values.

8. The method of claim 1, further comprising updating a flag based upon a result of the execution of the at least two of a plurality of microinstructions.

9. The method of claim 1, further comprising,

(a) if an unmasked exception occurs, canceling the execution of the microinstructions and invoking a microcode handler,

(b) if an unmasked exception does not occur, updating at least one exception flag by independently generating a logical OR of exceptions for a plurality of functional units.

10. A method for processing software instructions comprising,

(a) providing two microinstructions to emulate a high-half and a low-half SSE operation,

(b) forcing the high-half and low-half operations to issue in parallel,

(c) dispatching the high-half and low-half operations simultaneously to a first FP unit and to a second FP unit, respectively,

(d) generating a signal from an emulator's hardware,

(e) sending the signal to the first and second FP functional units,

(f) determining whether an exception is taken in either the first or the second FP unit,

(g) if an exception is taken in either the first or second FP unit, flushing a result in the other FP unit, and

(h) updating MXCSR flags based upon the results of the first and second FP units.

11. The method of claim 10, wherein the flushing of a result in the other FP unit does not depend upon the relative ages of the two microinstructions.

12. A computer system comprising,

a processor comprising,

(a) a floating point unit;

(b) a ROM;

(c) a plurality of floating point registers;

wherein the processor is configured to emulate an instruction set by:

(a) decomposing a macroinstruction into a plurality of microinstructions;

(b) issuing at least two of the plurality of microinstructions in parallel,

(c) determining whether an exception occurs in any of the at least two of a plurality of microinstructions, and

(d) if an exception occurs in any of the at least two of a plurality of microinstructions, canceling the at least two of a plurality of microinstructions.

13. The method of claim 12, further comprising executing at least two of the plurality of microinstructions.

14. The method of claim 13, wherein the least two of a plurality of microinstructions are executed on separate execution units, but appear as though they were executed on a single execution unit.

15. The computer system of claim 14, wherein the processor is further configured to emulate an instruction set by updating a flag based upon a result of the execution of the at least two of the plurality of microinstructions.

16. The computer system of claim 15, wherein the processor is further configured to emulated an instruction set by

(a) determining whether an exception occurs in the execution of any of the at least two of a plurality of microinstructions,

(b) if an exception occurs, causing the exception to cancel all of the at least two of a plurality of microinstructions.

17. The computer system of claim 12, wherein the instruction set is a SSE instruction set.

18. The computer system of claim 17, further comprising an FP register having 82 bits, wherein the computer system uses two FP registers to emulate four 32-bit single-precision, floating point values in an SSE register.

ABSTRACT

The present invention is a method for processing instructions by decomposing a macroinstruction into at least two microinstructions, executing the microinstructions in parallel, and linking the microinstructions such that they appear as though they were executed as a single functional unit. The present invention operates by determining whether certain exceptions occur in either of the functional units, according to SSE rules for exceptions. If an exception does occur in any of the linked microinstructions, then the execution of each of those microinstructions is canceled. This avoids the necessity of a back-off or undo mechanism.

Figure 1A

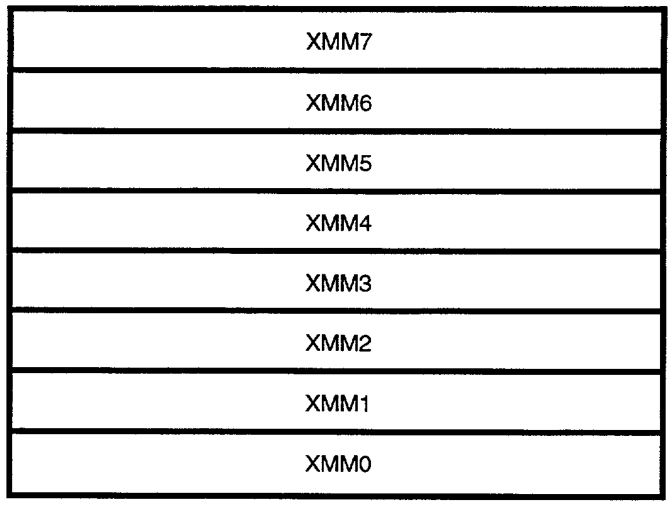


Figure 1B

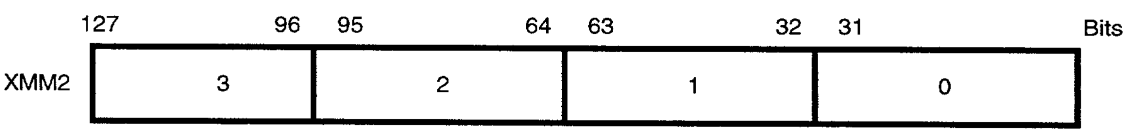


Figure 1C

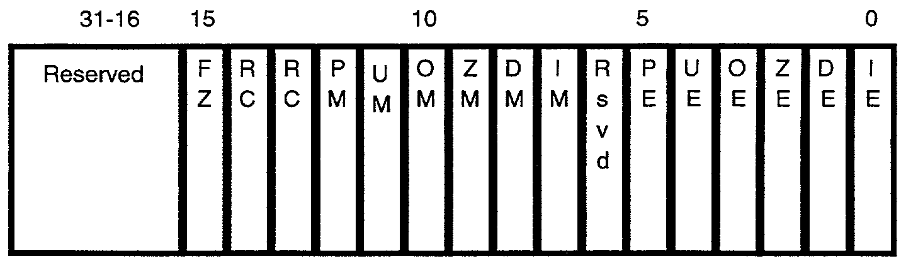
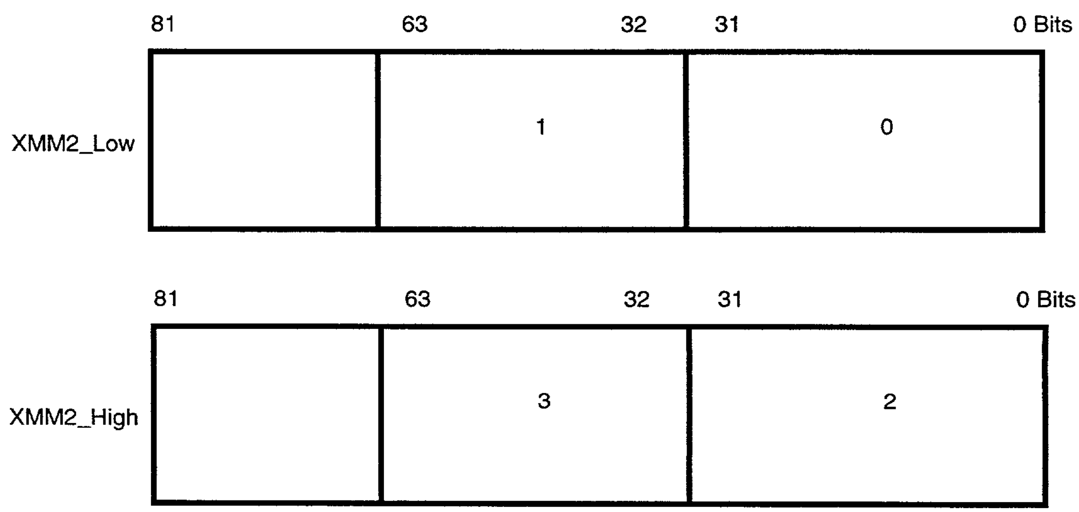


Figure 1D



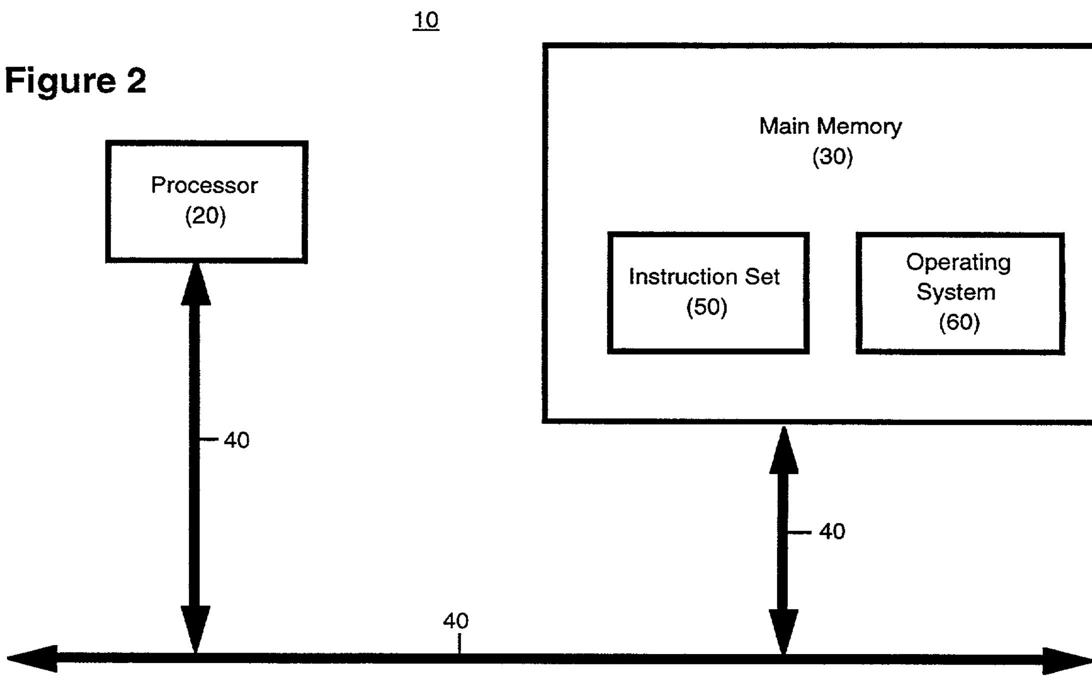

10
Main Memory
(30)
Processor
(20)
Instruction Set
(50)
Operating
System
(60)
40
40
40

Figure 2

Figure 3

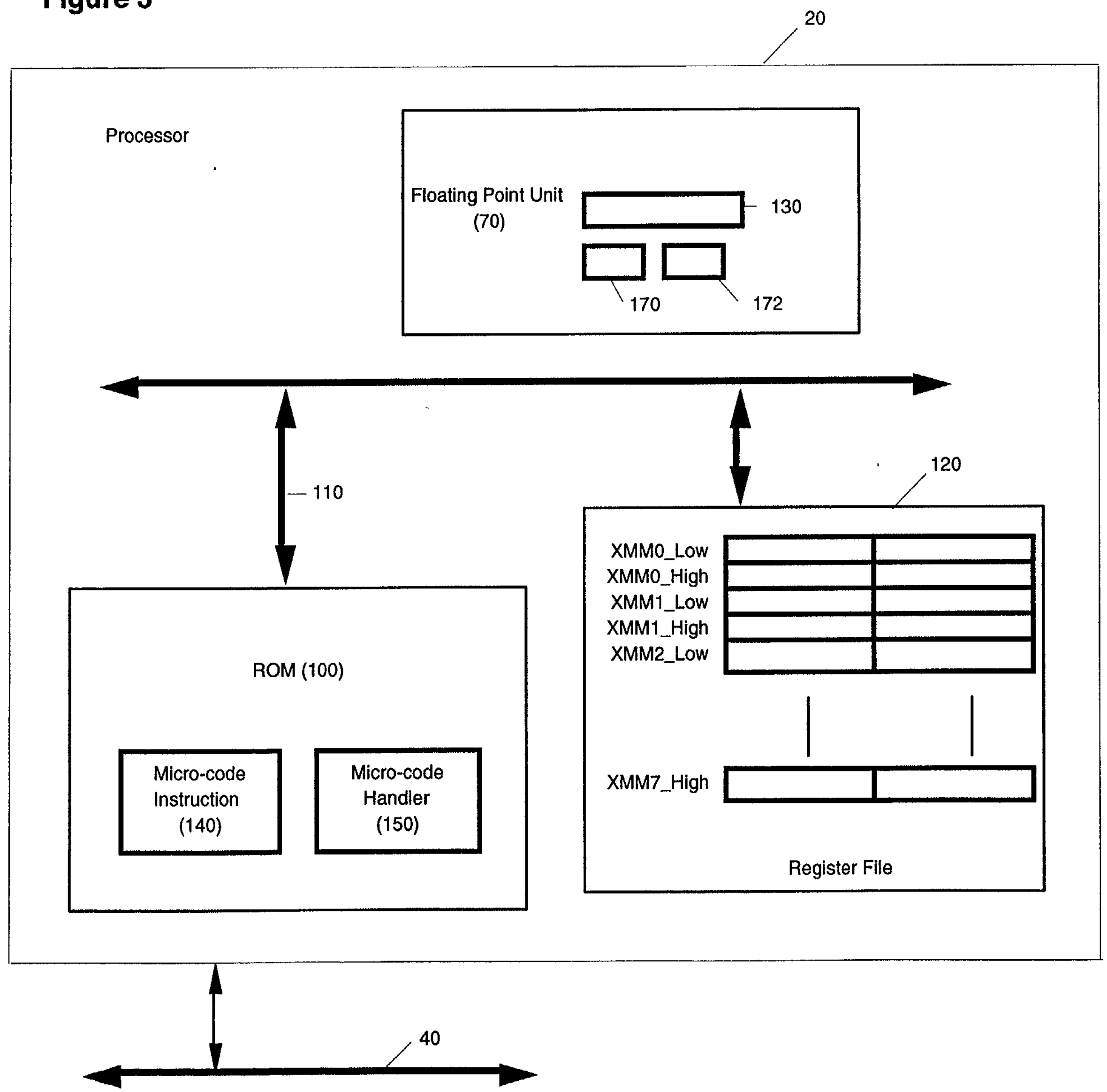
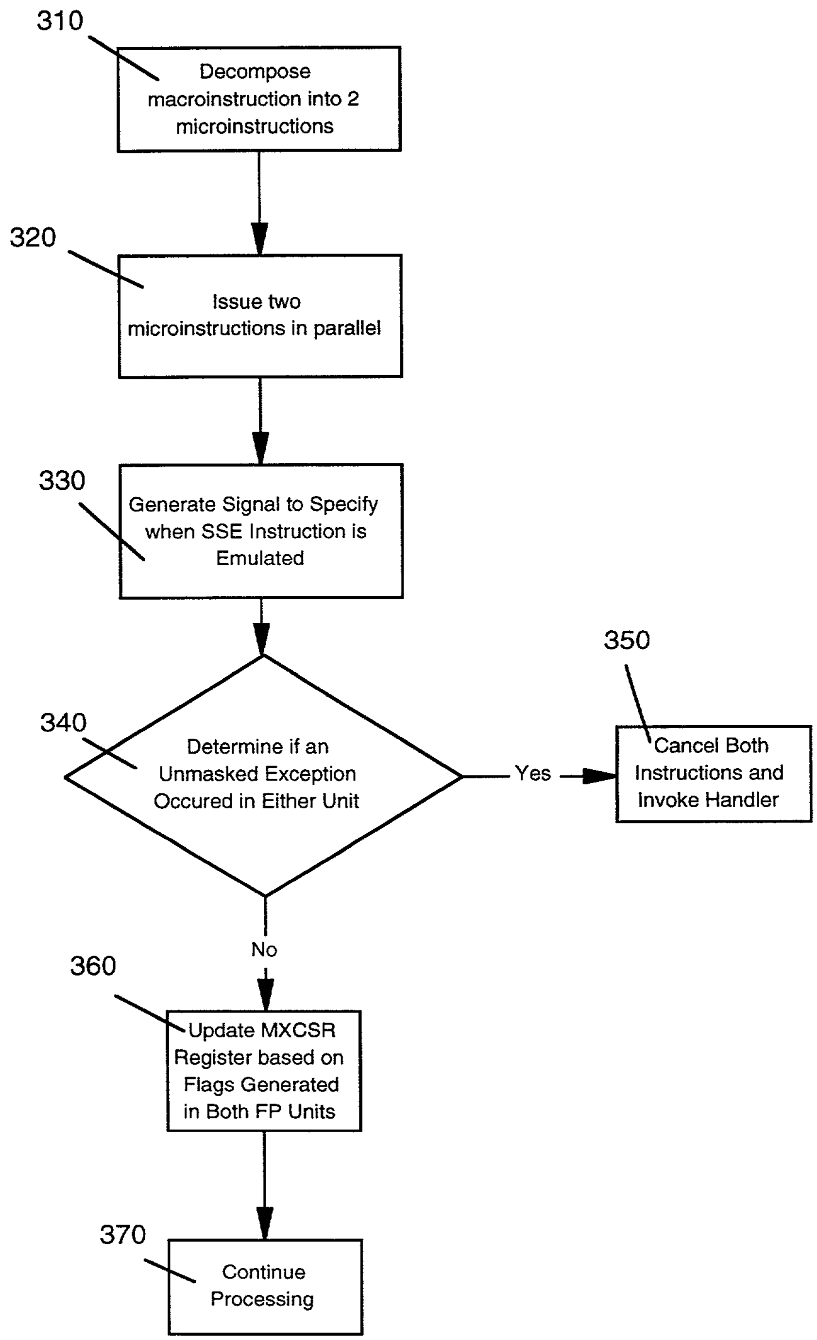

20
Processor
Floating Point Unit
(70)
130
170
172
110
120
ROM (100)
Micro-code
Instruction
(140)
Micro-code
Handler
(150)
XMM0_Low
XMM0_High
XMM1_Low
XMM1_High
XMM2_Low
XMM7_High
Register File
40

Figure 4


310
Decompose macroinstruction into 2 microinstructions
320
Issue two microinstructions in parallel
330
Generate Signal to Specify when SSE Instruction is Emulated
340
Determine if an Unmasked Exception Occured in Either Unit
Yes
350
Cancel Both Instructions and Invoke Handler
No
360
Update MXCSR Register based on Flags Generated in Both FP Units
370
Continue Processing

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION	ATTORNEY DOCKET NO. 10971393-1

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

Method And Apparatus For Emulating SIMD Instructions

the specification of which is attached hereto unless the following box is checked:

( ) was filed on ________________ as US Application Serial No. or PCT International Application Number ______________ and was amended on __________________ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
N/A			YES: ___ NO: ___
			YES: ___ NO: ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
N/A	

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)
N/A		

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Customer Number 022879

Place Customer Number Bar Code Label here

Send Correspondence to:
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80528-9599

Direct Telephone Calls To:
Alexander J Neudeck
(970) 898-4931

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Patrick Knebel **Citizenship:** US

Residence: 2201 Greenmont Ct Ft Collins CO 80524

Post Office Address: Same as residence

Inventor's Signature **Date** 1/31/00

Rev 11/99 (DecPwr) (Use Page Two For Additional Inventor(s) Signature(s))

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION (continued)

ATTORNEY DOCKET NO. 10971393-1

Full Name of # 2 joint inventor: Kevin David Safford Citizenship: US

Residence: 4100 Suncrest Drive Fort Collins CO 80525

Post Office Address: Same as residence

Inventor's Signature: Kevin David Safford Date: 1/28/2000

Full Name of # 3 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 4 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 5 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 6 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 7 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 8 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Rev 11/99 (DecPwr)

(Use Page Two For Additional Inventor(s) Signature(s))